Direct Link Communication I: Basic Techniques

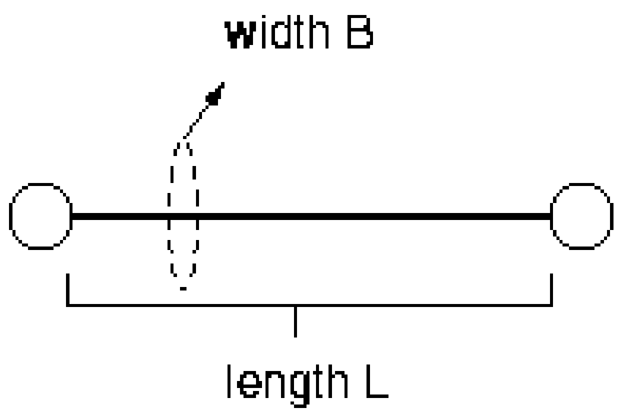
Data Transmission

Link speed unit: bps

$\longrightarrow$ abstraction

$\longrightarrow$ ignore carrier frequency, coding etc.

Point-to-point link:



$\longrightarrow$ wired or wireless

$\longrightarrow$ includes broadcast case

Interested in *completion time*:

$\longrightarrow$ time elapsed between sending/receiving first bit

- Single bit:

  $\rightarrow \approx L/\mathrm{SOL}$ (lower bound)

  $\rightarrow$ latency (or propagation delay)

  $\rightarrow$ optical fiber, wireless: exact

- Multiple, say S, bits:

  $\rightarrow \approx L/\mathrm{SOL} + S/B$

  $\rightarrow$ latency + transmission time

Latency vs. transmission time: which dominates?

$\longrightarrow$ a lot to send, a little to send, . . .

$\longrightarrow$ satellite, Zigbee, WLAN, broadband WAN

Reliable Transmission

Principal methodology: ARQ (Automatic Repeat reQuest)

$\longrightarrow$ use retransmission

$\longrightarrow$ used in both wired/wireless

- function duplication

  $\rightarrow$ link layer, transport layer, etc.

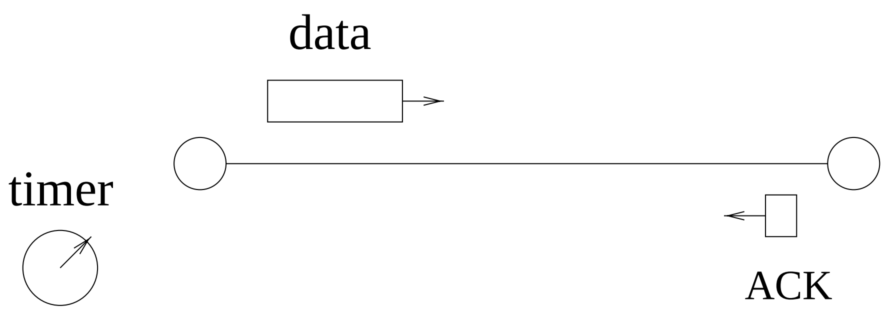
- alternative: FEC

  $\rightarrow$ not assured

  $\rightarrow$ hybrid schemes

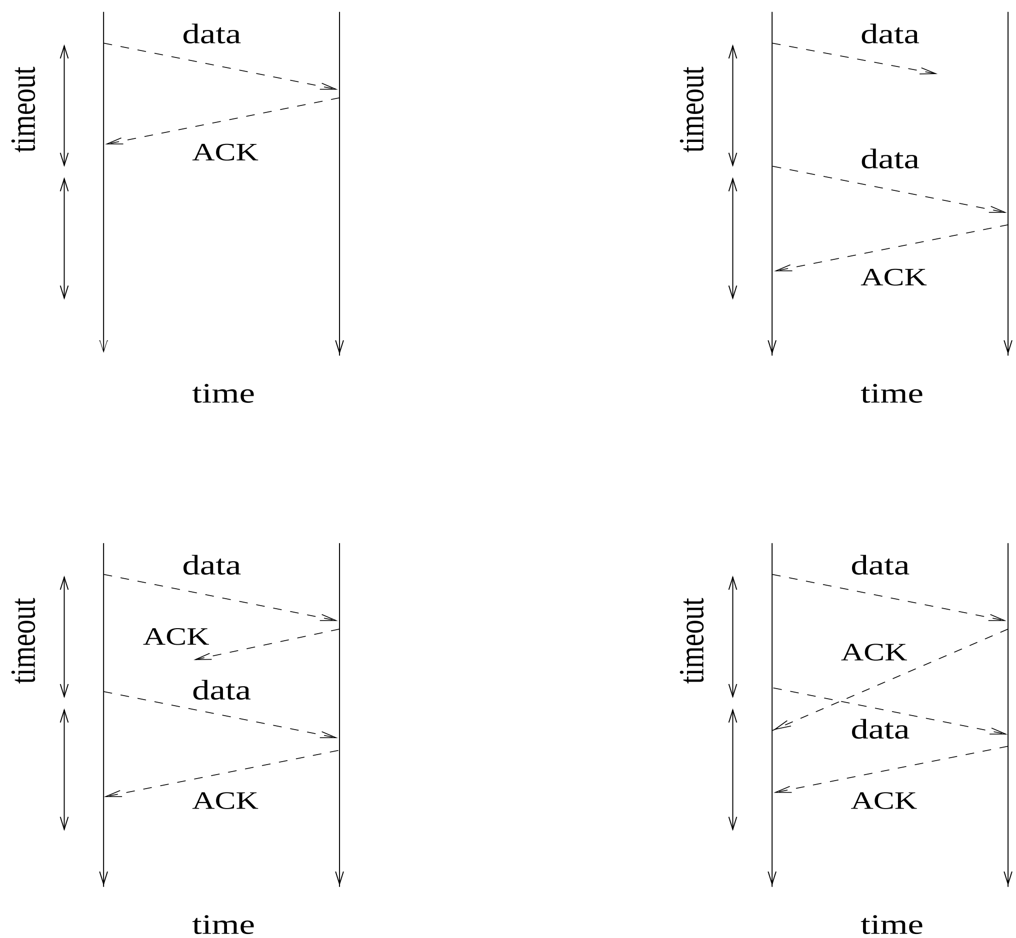
Three components:

- timer
- acknowledgment (ACK)
- retransmit



Stop-and-Wait

Assumption: Frame is "lost" due to corruption; discarded by NIC after error detection.



Issue of RTT (Round-Trip Time) & timer management:

- what is proper value of timer?

  → RTT estimation

- easier for single link

  → RTT is more well-behaved

- more difficult for multi-hop path in internetwork

  → latency + queueing effect

Another key problem: not keeping the pipe full.

$\longrightarrow$ delay-bandwidth product

$\longrightarrow$ volume of data travelling on the link

High throughput: want to keep the pipe full

Stop-and-wait throughput (bps):

- RTT
- frame size (bits)

$\longrightarrow$ throughput = frame size / RTT

Ex.: Link BW 1.5 Mbps, 45 ms RTT

- delay-bandwidth product:

  $\rightarrow$ 1.5 Mbps $\times$ 45 ms = 67.5 kb $\approx$ 8 kB

- if frame size 1 kB, then throughput:

  $\rightarrow$ $1024 \times 8/0.045 = 182$ kbps

  $\rightarrow$ utilization: only 182 kbps/1500 kbps = 0.121

Solution: increase frame size

- brute increase of frame size can be problematic

  $\rightarrow$ bully problem

  $\rightarrow$ existing LAN frame standards (legacy compatible)

- send blocks of data, i.e., sequence of frames

Sliding Window Protocol

⟶ send window/block of data

Issues:

- Shield application process from reliability management chore

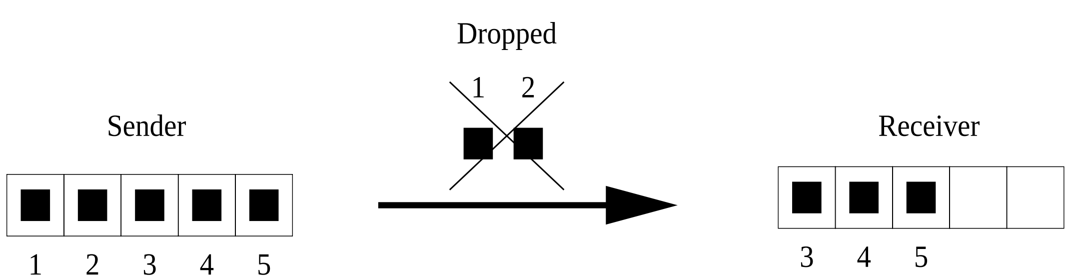
  → exported semantics: continuous byte stream

  → simple app abstraction: e.g., `read` system call

- Both sender and receiver have limited buffer capacity

  → efficiency: space-bounded computation

  → task: "plug holes & flush"



Simple solution when receiver has infinite buffer capacity:

- sender keeps sending at maximum speed
- receiver informs sender of holes

  → i.e., negative ACK
- sender retransmits missing frames

  ⟶ sender's buffer capacity?

  ⟶ need for positive ACK?

With finite buffer:

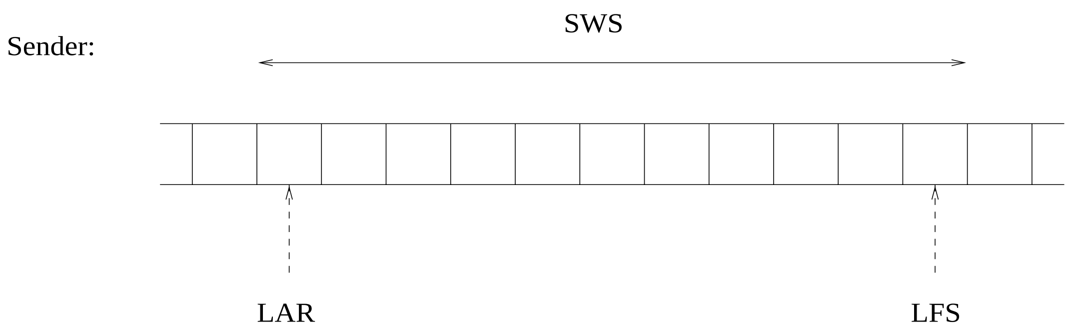
⟶ issue of bookkeeping

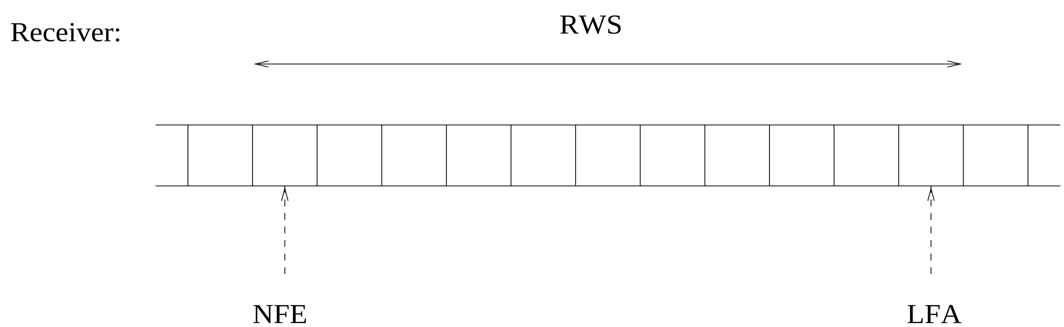
Flow control & congestion control:

→ sending too much is counterproductive

→ regulate sending rate

Set-up:





- *SWS*: Sender Window Size (sender buffer size)
- *RWS*: Receiver Window Size (receiver buffer size)
- *LAR*: Last ACK Received
- *LFS*: Last Frame Sent
- *NFE*: Next Frame Expected
- *LFA*: Last Frame Acceptable

Assign sequence numbers to frames.

$\longrightarrow$ IDs

Maintain invariants:

- $\text{LFA} - \text{NFE} + 1 \leq \text{RWS}$
- $\text{LFS} - \text{LAR} + 1 \leq \text{SWS}$

Sender:

- Receive ACK with sequence number X
- Forwind LAR to X
- Flush buffer up to (but not including) LAR
- Send up to $\text{SWS} - (\text{LFS} - \text{LAR} + 1)$ frames
- Update LFS

Receiver:

- Receive packet with sequence number Y
- Forwind to (new) first hole & update NFE

  $\longrightarrow$ NFE need not be $Y + 1$
- Send cumulative ACK (i.e., NFE)
- Flush buffer up to (but not including) NFE to application
- Update LFA $\leftarrow$ NFE + RWS $-$ 1

ACK variants:

- piggyback
- negative ACKs
- selective ACKs

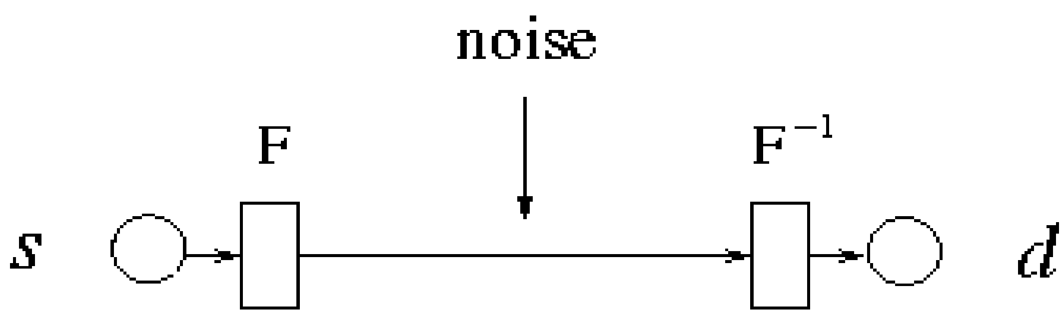
Sequence number wrap-around problem:

$$\text{SWS} < (\text{MaxSeqNum} + 1)/2.$$

$\longrightarrow$ note: stop-and-wait is special binary case

Error Detection and Correction

⟶ recall: reliable transmission over noisy channel



Key problem:

- sender wishes to send a; transmits code word w_a
- receiver receives w
- due to noise, w may, or may not, be equal to w_a

⟶ would like to detect error has occurred

⟶ would like to correct error

Error detection problem:

- determine if w is a valid code word

  $\longrightarrow$ i.e., for some symbol $c \in \Sigma$, $F(c) = w$

- e.g., parity bit in ASCII transmission

  $\longrightarrow$ odd or even parity

  $\longrightarrow$ limitation?

Error correction problem:

- even if $w \neq w_a$, recover symbol a from scrambled w

  $\longrightarrow$ correction is tougher than detection

- how to correct single errors for ASCII transmission?

  $\longrightarrow$ e.g., assume 21 bits available

  $\longrightarrow$ what about 14 bits?

Conceptual approach to detection & correction:

Error detection:

- valid/legal code word set $S = \{w_a : a \in \Sigma\}$
- can detect k-bit errors if

  $\rightarrow$ corrupted w does not belong to S

  $\rightarrow$ for all k-bit error patterns

  $\longrightarrow$ flipped code word cannot impersonate as valid

What kind of S can satisfy these properties?

$\longrightarrow$ e.g., ASCII with 1-bit, 2-bit, . . ., k-bit flips

$\longrightarrow$ intuition?

Key idea:

$\longrightarrow$ valid code words should not look alike

$\longrightarrow$ well-separatedness

$\longrightarrow$ "distance" between two binary strings?

Error correction:

- suppose w_a has turned into w under k-bit errors
- for all $b \in \Sigma$, calculate $d(w_b, w)$

  $\rightarrow$ use Hamming distance

  $\rightarrow$ e.g., $d(1011, 1101) = 2$
- pick $c \in \Sigma$ with smallest $d(w_c, w)$ as answer

Ex.: $0 \mapsto 000$ and $1 \mapsto 111$

$\longrightarrow$ want to send 0, hence send 000

$\longrightarrow$ 010 arrives: $d(010, 000) = 1$ & $d(010, 111) = 2$

$\longrightarrow$ conclude 000 was corrupted into 010

$\longrightarrow$ original data bit: 0

Obviously not fool-proof . . .

$\longrightarrow$ the larger k, the more distant the code words

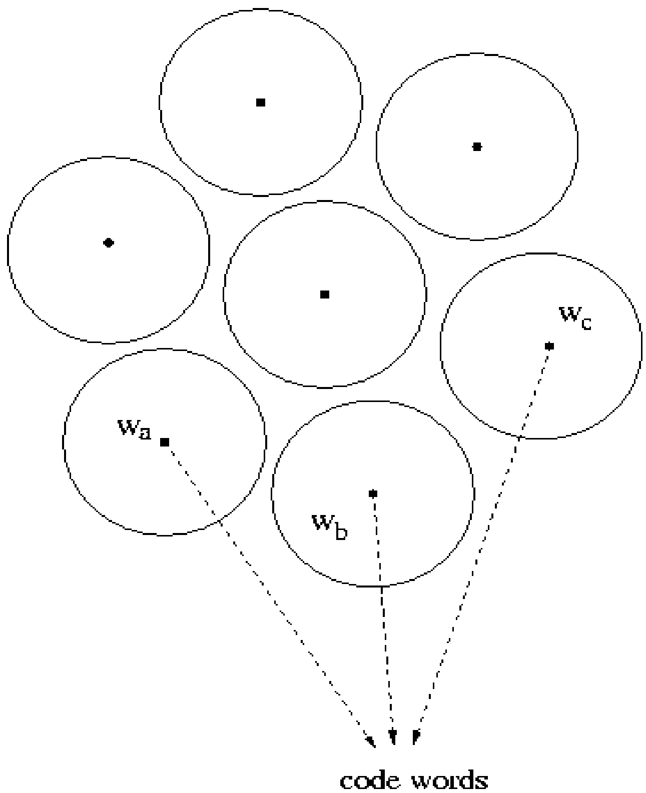
$\longrightarrow$ need a roomier playing area

$\longrightarrow$ imbed valid/legal code words

Pictorially: "ball" of radius r centered at w_a

$\longrightarrow$ $B_r(w_a) = \{w : d(w_a, w) \leq r\}$

$\longrightarrow$ well-separated code word set S layout



If k bit flips, sufficient conditions for error detection and correction in terms of $d(w_a, w_b)$ for all $a, b \in \Sigma$?

Network protocol context: different approach to detection vs. correction

⟶ error detection: use checksum and CRC codes

⟶ error correction: use retransmission

⟶ humans?

⟶ can also use FEC; for real-time data

Internet checksum: group message into 16-bit words; calculate their sum in one's complement; append "checksum" to message.

⟶ problem?